DEPARTMENT OF APPLIED ECONOMICS

RESEARCH REPORT

HEURISTIC BRANCH-AND-PRICE FOR BUILDING LONG TERM TRAINEE SCHEDULES

JEROEN BELIËN • ERIK DEMEULEMEESTER

OR 0422

Heuristic branch-and-price for building long term trainee schedules

Jeroen Beliën
Erik Demeulemeester

Katholieke Universiteit Leuven
Naamsestraat 69, 3000 Leuven
e-mail: jeroen.belien@econ.kuleuven.ac.be

Abstract

Branch-and-price is an increasingly important technique for solving large integer programming models. Staff scheduling has been a particularly fruitful area since these problems typically exhibit a decomposable structure. Beside computational efficiency branch-and-price produces two other important advantages in comparison with pure integer programming. Firstly, it often allows for a more accurate problem statement since many constraints which are hard to formulate in the integer program could be easily incorporated in the column generator. Secondly, a branch-and-price algorithm can easily be turned into an effective heuristic when optimality is no major concern. We illustrate these advantages for a medical trainee scheduling problem encountered at *Oogziekenhuis Gasthuisberg Leuven* and present some computational results together with implementation issues.

Keywords: Heuristic, branch-and-price, staff scheduling

1 Introduction

Operations research literature is replete with examples of integer linear programming (ILP) techniques being applied to staff scheduling problems (see e.g. Warner (1976) or Beaumont (1997)). ILP essentially involves the construction of a branch-and-bound tree in which bounds are calculated by linear program (LP) relaxations and branching occurs by fixing variables having a fractional value to integer values. Certain classes of ILP problems exhibit a specific structure making them suitable to be decomposed into master and subproblems. Staff scheduling problems could for instance be seen as the task to schedule a number of staff members (each one representing a subproblem) such that a number of work requirements are satisfied over the time horizon (master problem). Decomposition involves a reformulation of the problem which often has far more variables than the original ILP. It is well known that a decomposition approach entails several advantages compared to pure ILP

branch-and-bound (Barnhart et al (1998)). First of all, computation times could be decreased dramatically since the LP relaxation of the reformulated problem is typically much tighter. Secondly, decomposition often allows for a larger modeling power since many constraints which are difficult or impossible to state in the original ILP formulation could be easily incorporated in the column generator (the dedicated algorithm which solves the subproblem). This is especially true if the decomposition approach has a natural interpretation which is the case in staff scheduling problems. A last important advantage is that branch-and-price algorithms could easily be turned into heuristic algorithms. During the last decade branch-and-price has gained considerable attention for solving staff scheduling problems, since these problems often exhibit a decomposable structure. Most of the encountered scheduling problems studied in the literature are short-term shift scheduling problems involving some kind of set covering or set partitioning formulation (e.g. Mason and Smith (1998), Mehrotra et al (2000), Caprara et al. (2003)). Alternatively, 0-1 multi-commodity flow formulations are proposed (e.g. Cappanera and Gallo (2001), Beliën and Demeulemeester (2004a)). The objective of this paper is threefold. First of all, it illustrates the advantages of a branch-and-price approach by means of a concrete example. It is shown how a number of *difficult to state* constraints are easily incorporated in the column generator and how both master and column generator could be made heuristically. Secondly, implementation issues and computational results are given for some real life instances for a trainee scheduling problem encountered at the *Oogziekenhuis Gasthuisberg Leuven.* Thirdly, a graphical user interface (GUI) is presented which allows for easy data input, constraint specification, visualization of the search process and visualization (and modification) of the schedule found by the algorithm.
The paper is organized as follows. In Section 2 the problem will be stated and written as an integer program. In Section 3 the problem is decomposed into master and subproblems and a branch-and-price algorithm will be elaborated. Section 4 discusses several heuristic extensions of the branch-and-price scheme. Section 5 presents the GUI developed for this application. Finally, Section 6 provides some computational results.

2 Problem Statement

The problem addressed in this paper involves the construction of 1-year trainee schedules at a hospital department and is a generalization of the problem dealt with in Beliën and Demeulemeester (2004a). Trainees are graduate medical science students who wish to specialize further in a specific field of health care. In order to complete their education they have to follow a number of trainee posts for a number of years. Trainee schedules are build at the start of each academic year. Such a schedule defines for each week of the year for each trainee which activity (s)he will perform during that week. The construction of these schedules is often a complicated task. First of all, the trainees make up an important human resource for performing

a variety of activities. Consequently, a number of coverage constraints must be taken into account when building the schedule. Since there is a large difference between for instance a first year trainee and a fourth year trainee, most of these coverage constraints are skill dependent. Secondly, a lot of constraints apply on the individual trainee level. The trainee schedules have to satisfy a number of aggregate formation requirements and trainees are not always available to be scheduled due to weeks off or trips abroad. Finally, since certain activities require a significant mastering time, these activities are best performed in a consecutive way by each trainee rather than splitting them up in several parts.

Due to all these constraints, the development of trainee schedules is an extremely complicated task. In order to provide more insight into the problem, we will shortly describe how this task was done up till now. In a first step, the responsible scheduler collects the required data. Coverage constraints and formation requirements are provided by the head of the department. Non-availability constraints are collected in a hierarchical way. A list circulates in which the trainees successively indicate during which weeks they will be absent and during which weeks they would like to take vacation. To ensure that vacation periods are sufficiently spread, the number of trainees having vacation at the same time is limited. Next, using pencil and paper, the scheduler tries to find a schedule that satisfies as many constraints as possible. Hereby, she mainly concentrates on the satisfaction of the coverage constraints. At certain moment, typically when 75% of the schedule is completed, she fails to satisfy the next coverage constraints without violating one or more of the formation, non-availability or setup constraints. At that moment, she tries to solve the schedule conflict by making a number of assignments undone or performing a number of switches. If she fails to solve the conflict in a limited number of tries, she accepts the violation of one (or more) constraints and continues construction. Upon completion, the schedule is communicated to all persons involved (trainees and surgeons). Since this task essentially involves the solution of a complex combinatorial puzzle and PC's are typically more suitable to solve such problems than humans, we believe that a well-thought-out algorithm could save construction time as well as generate qualitatively better schedules. First, we will state the problem mathematically as an integer program.

If we are to state an integer programming formulation for this problem, we first have to define a set of decision variables. Let i be the period index, j the trainee index and k the activity index. Since we have to decide for each trainee which activity (s)he will perform during each week, a natural choice of decision variables includes:

$$x_{ijk} = \begin{cases} 1, & \text{if during period } i \text{ trainee } j \text{ is scheduled to perform activity } k; \\ 0, & \text{otherwise.} \end{cases}$$

Let us now state the coverage, formation, non-availability and setup constraints in terms of these decision variables. Firstly, consider the coverage constraints. A coverage constraint applies on a specific time horizon and requires a particular activity

to be performed by a certain number of trainees during each period in the time horizon. These coverage constraints are often skill dependent. Therefore, a set of trainees having the appropriate skills is being defined for each coverage constraint. An example of a coverage constraint is as follows: "For each period starting from period 1 until period 16 exactly 2 trainees out of the set {trainee 1, trainee 2, trainee 5 and trainee 6} have to perform activity 1". For a particular activity, several coverage constraints could be specified, but each coverage constraint involves only one activity. Let C represent the set of coverage constraints and let a_c, T^c, R_c, s_c and e_c respectively be the activity, the trainee set having the appropriate skills, the number of trainees required and the start and end period of the time horizon of coverage constraint c. Then, each coverage constraint can be stated as follows:

$$\sum_{j \in T^c} x_{ija_c} = R_c \qquad \forall c \in C \text{ and } \forall i = s_c..e_c \tag{1}$$

Secondly, consider the formation requirements. Let n be the number of periods in the scheduling horizon, m the number of trainees and p the number of activities. If F_{jk} is the target number of periods trainee j has to perform activity k, the formation requirements are as follows:

$$\sum_{i=1}^{n} x_{ijk} = F_{jk} \qquad \forall j = 1..m \text{ and } \forall k = 1..p \tag{2}$$

Thirdly, if the set N^j contains all non-available periods for trainee j, the non-availability constraints are as follows:

$$\sum_{k=1}^{p} x_{ijk} = 0 \qquad \forall j = 1..m \text{ and } \forall i \in N^j \tag{3}$$

Obviously, a trainee cannot perform more than one activity during a certain period:

$$\sum_{k=1}^{p} x_{ijk} \leq 1 \qquad \forall i = 1..n \text{ and } \forall j = 1..m \tag{4}$$

Finally, to state the setup constraints one also needs the following 0-1 setup variables:

$$y_{ijk} = \begin{cases} 1, & \text{if trainee } j \text{ starts activity } k \text{ at period } i; \\ 0, & \text{otherwise.} \end{cases}$$

The setup constraints are as follows:

$$y_{1jk} = x_{1jk} \qquad \forall j = 1..m \text{ and } \forall k = 1..p \qquad (5)$$

$$y_{ijk} \geq x_{ijk} - x_{(i-1)jk} \qquad \forall i = 2..n \text{ and } \forall j = 1..m \text{ and } \forall k = 1..p \qquad (6)$$

$$\sum_{i=1}^{n} y_{ijk} \leq 1 \qquad \forall j = 1..m \text{ and } \forall k = 1..p \qquad (7)$$

Typically, these kinds of problems are *overconstrained*, meaning that no feasible solution can be found which satisfies all these constraints. Therefore, we introduce for each constraint a dummy variable except for constraint 4, since this constraint can, of course, not be violated. Since we wish to satisfy as many constraints as possible, the objective function of our ILP model is to minimize the total sum of these dummy variables. Because some constraints are more important than others each dummy could be weighted with a *penalty cost*. Let r_{ci}^- and r_{ci}^+ respectively be the number of trainees scheduled too few and the number of trainees scheduled too many in period i for coverage constraint c and p_c^- and p_c^+ their respective associated penalty costs (these costs are assumed to be constant over all periods of the coverage constraint horizon, however this is not required for our algorithm). Equivalently, let f_{jk}^- and f_{jk}^+ be the respective number of periods too few and too many (as compared to formation requirements) trainee j is scheduled to perform activity k and v_{jk}^- and v_{jk}^+ their associated penalty costs. Let L_{jk} and U_{jk} be the strict minimum and maximum number of periods trainee j has to perform activity k as stated in the formation requirements. d_{ij} is the dummy variable forced to be 1 if trainee j is scheduled to perform an activity at period i although actually not available and w_{ij} is the respective penalty cost. Finally, h_{jk} equals the number of restarts of activity k by trainee j and g_{jk} is the associated penalty cost. The ILP formulation is given below. The last constraint set (18) defines x and y as binary variables. Obviously, the dummy variables cannot be negative. Observe that this formulation produces $2 * n * m * p$ binary variables. Typical problems encountered in practice involve fifty periods, twenty trainees and ten activities. A number of tests revealed that commercial ILP solvers cannot solve the integer programming problem for these dimensions within reasonable time limits. This forces us to look for an alternative solution approach.

$$\text{Min} \sum_{c \in C} \sum_{i=s_c}^{e_c} (p_{ci}^+ r_{ci}^+ + p_{ci}^- r_{ci}^-) + \sum_{j=1}^{m} \sum_{k=1}^{p} (v_{jk}^- f_{jk}^- + v_{jk}^+ f_{jk}^+ + g_{jk} h_{jk}) + \sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij} d_{ij} \qquad (8)$$

Subject to:

$$\sum_{j \in T^c} x_{ija_c} - r_{ci}^{+} + r_{ci}^{-} = R_c \qquad \forall c \in C \text{ and } \forall i = s_c..e_c \qquad (9)$$

$$\sum_{i=1}^{n} x_{ijk} - f_{jk}^{+} + f_{jk}^{-} = F_{jk} \qquad \forall j = 1..m \text{ and } \forall k = 1..p \qquad (10)$$

$$\sum_{i=1}^{n} x_{ijk} \geq L_{jk} \qquad \forall j = 1..m \text{ and } \forall k = 1..p \qquad (11)$$

$$\sum_{i=1}^{n} x_{ijk} \leq U_{jk} \qquad \forall j = 1..m \text{ and } \forall k = 1..p \qquad (12)$$

$$\sum_{k=1}^{p} x_{ijk} - d_{ij} = 0 \qquad \forall j = 1..m \text{ and } \forall i \in N^j \qquad (13)$$

$$\sum_{k=1}^{p} x_{ijk} \leq 1 \qquad \forall i = 1..n \text{ and } \forall j = 1..m \qquad (14)$$

$$y_{1jk} = x_{1jk} \qquad \forall j = 1..m \text{ and } \forall k = 1..p \qquad (15)$$

$$y_{ijk} \geq x_{ijk} - x_{(i-1)jk} \qquad \forall i = 2..n \text{ and } \forall j = 1..m \text{ and } \forall k = 1..p \qquad (16)$$

$$\sum_{i=1}^{n} y_{ijk} - h_{jk} \leq 1 \qquad \forall j = 1..m \text{ and } \forall k = 1..p \qquad (17)$$

$$x_{ijk}, y_{ijk} \in \{0, 1\} \qquad \forall i = 1..n \text{ and } \forall j = 1..m \text{ and } \forall k = 1..p \qquad (18)$$

3 Decomposition of the problem

It is well known that staff scheduling problems can often easily be decomposed into a master and subproblem structure. To decompose (8)-(18) on the trainees, we introduce decision variables which represent individual trainee schedules. Let binary decision variable z_{jt} be defined as follows:

$$z_{jt} = \begin{cases} 1, & \text{if schedule } t \text{ was chosen for trainee } j; \\ 0, & \text{otherwise.} \end{cases}$$

z_{jt} represents a particular schedule t for a single trainee j and is defined by coefficients a_{ijkt}. a_{ijkt} equals 1 if in schedule t trainee j performs activity k during period i and equals 0 otherwise. Let c_{jt} be the total cost of schedule t for trainee j (thus including all trainee-specific penalty costs stated earlier) and NC_j the total number of different schedules for trainee j. The model can then be rewritten as follows:

$$\text{Minimize} \sum_{j=1}^{m} \sum_{t=1}^{NC_j} c_{jt} z_{jt} + \sum_{c \in C} \sum_{i=s_c}^{e_c} (p_{ci}^{+} r_{ci}^{+} + p_{ci}^{-} r_{ci}^{-}) \qquad (19)$$

Subject to:

$$\sum_{j \in T^c} \sum_{t=1}^{NC_j} a_{ijkt} z_{jt} - r_{ci}^{+} + r_{ci}^{-} = R_c \qquad \forall c \in C \text{ and } \forall i = s_c..e_c \qquad (20)$$

$$\sum_{t=1}^{NC_j} z_{jt} = 1 \qquad \forall j = 1..m \qquad (21)$$

$$z_{jt} \in \{0, 1\} \qquad \forall j = 1..m \text{ and } \forall t = 1..NC_j \qquad (22)$$

The objective function (19) is again the minimization of costs, but now expressed in terms of the new z_{jt} variables. Observe that the coverage constraints (20) are the only constraints taken over from the original ILP model (9)-(18). Constraints (10)-(17) and (18) of the original formulation will be implicitly satisfied within each column. Additionally, constraint set (14) of the old formulation will be satisfied in the new formulation by constraint set (21) which implies that exactly one schedule has to be selected for each trainee. This decomposition has a natural interpretation. If the total schedule is visualized by a matrix in which the rows represent the time periods and the columns represent the trainees, each z_{jt} corresponds to a particular implementation of column j in this matrix.
The main advantage of the new formulation compared to the original formulation is that the LP relaxation of the new model is much higher than that of the original model and hence gives a better approximation of the best possible integral solution. Consequently, the branch-and-bound tree remains within reasonable proportions. An important drawback of the new model is that it can have far more variables than can be reasonably attacked directly. It is however not necessary to enumerate all possible columns to solve the LP to optimality. The LP can be solved by using only a subset of the columns and can generate more columns as needed. This way of LP optimizing is called column generation. We iteratively add new columns and solve the restricted model until no more columns can be found which could decrease the objective function further. The model depicted in (19)-(21) is referred to as the master problem. The master is called restricted if it does not contain all possible columns. The problem of finding the next most improving column for each trainee j is called the subproblem or pricing problem. The information provided by the dual prices of constraints (20) and (21) obtained after each solution of the restricted master can be used to determine whether the master is solved to optimality. This will be the case if no more columns can be found with negative reduce cost. Let π_{ic} represent the dual prices of restrictions (20) and let μ_j represent the dual prices of restrictions (21). The reduced cost of a new column t for trainee j is then given by:

$$\mu_j + c_{jt} + \sum_{c \in C | j \in T^c} \sum_{i=s_c}^{e_c} \pi_{ic} a_{ija_c t} \qquad (23)$$

Hence, finding the next most improving column for trainee j corresponds to finding the column that minimizes (23). When no more columns can be found with negative reduced cost, the master LP is solved to optimality. If this solution contains fractional values (i.e. z_{jt}'s with value between 0 and 1), we need a way of enforcing integrality in order to obtain a feasible schedule. A branch-and-bound enumeration scheme in which bounds are calculated by LP relaxation and the LP is solved through column generation, is called a branch-and-price algorithm. Summarizing, to solve our problem we reformulated it such that the decision variables represent individual trainee schedules. Since this reformulation involves a huge number of variables, a branch-and-price framework is used to solve it. Such an approach requires two major pieces: a method for generating the 'best' trainee schedule relative to our current set and a method for branching away from fractional solutions. These two pieces must be coordinated, since a choice of one implies limitations on the choice of the other. In what follows we will briefly elaborate on both pieces.

3.1 Column generation

The generation of the next best column for a trainee requires the solution of a subproblem or pricing problem. This can be thought of as finding the cheapest path through a network in which the different arc costs constitute of the individual trainee penalty costs on the one hand and the dual prices of the coverage constraints to which the activity-trainee assignments make a contribution on the other hand. Let us illustrate this by means of a simple example. Suppose we are searching a new column for trainee j. Given is a matrix of costs. The columns of this matrix represent all activities which could be performed by trainee j. Observe that there is always the possibility that a trainee performs no activity during a certain time period. Therefore, one column has to be included which represents performing 'no activity'. The rows represent the time horizon. Each cell of the matrix has a cost q_{ik} which is the sum of the corresponding non-availability cost p_{ij} on the one hand and the sum of contributing dual prices π_{ic} on the other hand. This matrix has to be traversed from top to bottom in the cheapest possible way, while visiting each column (except the last one) between a minimum and a maximum number of rows. Table 1 represents an instance of such a pricing problem for a trainee who has to perform three activities all preferably for three periods. Each unit deviation of this target number produces a penalty cost of 2, however deviations larger than one period are not allowed. In other words, each activity has to be performed between two and four periods. Moreover, assume for the moment that activity split-ups are not allowed.

Table 1: Pricing problem for trainee j: a feasible path is indicated in bold.

	[a] $q_{ik} = p_{ij} + \sum_{c \in C \mid j \in T^c \text{ and } a_c = k \text{ and } s_c \leq i \leq e_c} \pi_{ic}$			
Period i	Activity $k = 1$	Activity $k = 2$	Activity $k = 3$	No activity
1	**-2**	-1	1	0
2	**-2**	1	1	0
3	**-3**	2	3	0
4	2	1	**-1**	0
5	0	2	**-2**	0
6	1	5	3	**0**
7	4	2	7	**0**
8	-2	4	-6	**0**
9	0	**-4**	8	0
10	1	**-1**	3	0

[a] For ease of explanation all cost values are integer. Note however that during column generation these cost values are usually fractional due to the dual prices.

Table 1 shows a feasible path through this matrix in bold. It has a total price of -11 = -7 (scheduling activity 1 from period 1 to 3) + (-3) (scheduling activity 3 from period 4 to 5) + 2 (one time unit too few for activity 3) + (-5) (scheduling activity 2 from period 9 to 10) + 2 (one time unit too few for activity 2). Hence, if the dual price μ_j is smaller than 11, this column will be added to the master. Figure 1 visualizes part of the explored network for this example. Hereby, the nodes represent states which are defined by (a) the reached time period (x-axis) and (b) the set of scheduled activities (y-axis). For ease of representation only those arcs are drawn that indicate the transitions of the path found in Table 1.

The pricing problem could be solved using a recursive algorithm which (implicitly) explores the whole state space of feasible paths. Hereby, the construction of a path can be pruned based on (a) dominance rules or (b) bound comparisons. A simple dominance rule is as follows. If a certain state has already been reached by a previously explored path at a lower cost, then the construction of the current path can be pruned, since it will never result in a better path. A lower bound on the remaining costs can easily be calculated for each state as the summation of the costs of all best possible assignments of the not yet scheduled activities. If during the construction of a certain path a state is reached for which its current cost augmented with this lower bound exceeds an upper bound, then the construction of this path can also be pruned. Since columns with a higher cost than $-\mu_j$ have a positive reduced cost, the upper bound can initially be set to $-\mu_j$. Obviously, the upper bound is decreased each time the algorithm finds a lower cost path. Both pruning rules dramatically decrease the required solution times for solving the pricing problem. This approach works well for the problem dimensions we have considered (no trainee has to perform more than eight different activities) and if activity split-ups are prohibited. If activity split-ups have to be taken into account, the state space grows exponentially since it requires the introduction of new states

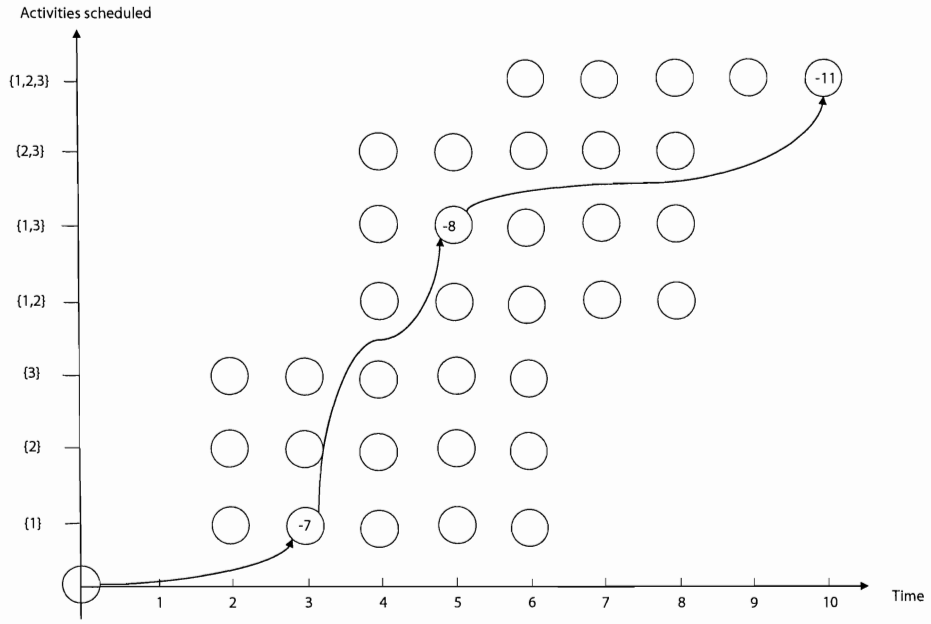


Figure 1: A part of the explored network

which track the number of periods each activity is scheduled instead of merely keeping track whether or not a certain activity is scheduled. Therefore, instead of an exact dynamic programming approach, an approximation algorithm could be applied for pricing out new columns (see Section 4). Per master optimization exactly one pricing problem is solved for each trainee. Hence, the dual prices are updated after the addition of at most m columns.

An important advantage of the decomposition approach is that it often allows for a larger modeling power, since many constraints which are difficult or impossible to state in the original ILP formulation, could be easily incorporated in the column generator. Suppose for instance that we wish to extend our model with precedence constraints. Assume that a trainee j cannot perform a certain activity k' before (s)he has already performed another activity k. In the original ILP formulation this would require for the inclusion of many additional constraints (i.e. $x_{1jk'} = 0$, $x_{2jk'} \leq x_{1jk}$, $x_{3jk'} \leq x_{1jk} + x_{2jk}$, ...). In the decomposition approach it suffices to simply exclude certain arcs from the network of the column generator.

The column generator thus takes care of all constraints but the coverage constraints. Since the number of coverage constraints dramatically exceeds the number of convexity constraints (only one for each trainee), this first set of constraints has a large impact on the computation times of the restricted masters. As a matter of fact, each extra coverage constraint generally tends to complicate the problem, whereas each extra trainee-specific constraint tends to simplify the problem, since it often results in smaller pricing problem networks and/or faster pruning in the column generator. Hence, each coverage constraint that could be transformed into one or more trainee-specific constraints may lead to a significant decrease in required computation time.

Therefore, we implemented some simple rules to identify such 'transformable' coverage constraints. Obviously, coverage constraints that apply on a single trainee can easily be transformed. Secondly, coverage constraints of capacity type (= or $\leq$) with right hand side value equal to zero can also easily be left out of the master. In the real-life problem we considered, first-year trainees were for instance not allowed to perform emergency-related activities during the first semester. Instead of keeping a constraint of the form $x_{11k} + x_{12k} + x_{13k} \cdots + x_{1mk} \leq 0$ for each period $1..\frac{n}{2}$ in the master, it is much more efficient to remove these constraints and imply them explicitly in the column generator by removing the corresponding arcs out of the networks. This is called constraint preprocessing. Since it reduces the complexity of both master and subproblems, constraint preprocessing is an important technique for decreasing overall computation times.

3.2 Branching

One difficulty in using column generation for the solution of integer programs is the development of branching rules to ensure integrality. Standard variable fixing is generally not a good idea for restricted integer programs where the columns are generated by implicit techniques. Consider for instance the rule of branching on a fractional variable, where the variable is set to 1 in the left branch and to 0 in the right branch. The subproblem in the left branch causes no problem, however the other subproblem is more difficult. Since it is possible (and quite likely) that the next best column to enter is precisely the one excluded by branching. Consequently, the column generator may be required to find the 2^{nd}, 3^{rd} or l^{th} best column in level l of the branch-and-bound tree. When columns can be associated with paths in a network, a possible branching scheme consists of fixing single components of the arc incidence vector (Vanderbeck (2000)). If this branching principle is applied to our problem, it results in branching on the original x_{ijk} variables. The next x_{ijk} to branch on is found by first searching two fractional columns for the same trainee j. Then, we search the first time period i for which the scheduled activity differs in both columns. Suppose activity k is the activity scheduled during the conflicting time period in the first fractional column. Then, x_{ijk} is set to 1 in the left branch and to 0 in the right branch. It can easily be shown that this branching scheme is complete. In other words, upon detection of a fractional solution, it is always possible to find a pair of fractional columns to initiate a new branch. The main advantage of this branching scheme is that it does not destroy the structure of the pricing problem, because the resulting modifications simply entail amending the cost of the corresponding arc in the underlying network. The timetable costs in the pricing problems are modified as follows. If x_{ijk} is set to 1, $c_{ik'}$ is set to $+\infty$ for all activities $k' \neq k$ in the pricing problem of trainee j. Else if x_{ijk} is set to 0, c_{ik} is set to $+\infty$ in the pricing problem of trainee j.
Since we have a method to generate columns and a branching scheme to cut away fractional solutions, our branch-and-price algorithm is complete. In order to ensure that the algorithm can deal with large problem dimensions, it is extended with a

number of heuristic features. These are discussed in the next section.

4 Heuristic extensions

Preliminary tests revealed that proven optimal solutions could only be found within reasonable time limits if either the total number of possible columns for each trainee is restricted (i.e. small problem dimensions, see Beliën and Demeulemeester (2004b)), or if only a small subset of columns has to be explicitly generated in order to find an optimal integral solution. The latter generally occurs if (a) the master LP relaxation is equal to the optimal IP solution value or (b) if the total set of feasible columns could be divided into two subsets, a 'cheap' set, containing a relative small number of cheap columns and an 'expensive' set containing all other columns, such that a feasible solution can be found with only columns from the 'cheap' set and the corresponding solution value is smaller than the cost of each column of the 'expensive' set. Consider for instance the case in which (part of) the setup costs are higher than the total cost of a feasible schedule. Then, the paths emerging from such an expensive split-up can immediately be pruned in the column generator. If, however, both (a) and (b) are not true, then the algorithm has to be extended with a number of heuristic features to ensure that at least a good (not necessarily optimal) solution will be found. We will successively deal with the following heuristic extensions:

- heuristic algorithm for pricing out new columns;
- premature termination of column generation;
- imbalanced branching;
- combining depth-first and best-first search;
- heuristically fixing x_{ijk} variables.

4.1 Heuristic algorithm for pricing out new columns

Thanks to the tremendous progress in LP optimization code the bottleneck of many branch-and-price implementations nowadays mostly lies in the solution of the pricing problems. As already mentioned in Section 3.1, the state space of the pricing problem explodes if activity split-ups are to be taken into account. Therefore, a heuristic algorithm has to be applied to price out new columns instead of an exact approach. Hereby, we make the following assumptions:

- each activity can only be restarted once for each trainee;
- if a trainee j restarts an activity k, or alternatively, if the activity is split up into two separate parts, then the first part has to totalize at least the minimum requirement L_{jk} of periods.

The first assumption can be justified since the setup penalty costs are usually much higher than the non-availability penalty costs and consequently activities that are started more than twice tend to occur rarely in (sub)optimal schedules. The second assumption can also be justified since it stands for a real-life constraint in many practical situations, namely that only an already experienced trainee can replace another trainee to perform an activity the latter cannot perform for one or two weeks. Moreover, it is not desirable that a trainee, who performs an activity for the first time, already quits it after having performed it for a relatively small number of periods. Analogous to precedence constraints, whereas this extra constraint can easily be dealt with in the column generator, it would have been very difficult to imply in the pure IP formulation.

Both assumptions entail two interesting properties. First of all, the total state space of feasible schedules (columns) is dramatically reduced for each trainee. Secondly, the dominance rule as well as the upper bound calculation stated above can still be applied. Nevertheless, generating the best column for certain trainees may still be (too) time consuming (recall that this has to be done many times). Observe, however, that it is not necessary to find optimal columns during early stages of column generation. Instead, good but not necessarily optimal columns, generated by a heuristic algorithm, can lower the master LP objective value already close to the optimal value. Therefore, the column generator outlined above will be truncated after the exploration of a limited number of feasible paths. At each new pricing iteration, the order in which the activities are considered is determined at random in order to ensure that all parts of the feasible path state space are more or less explored. If optimality proving would be the major concern, the column generator may not be truncated upon convergence of column generation, since only optimal columns can provide the information to determine whether or not the master objective value is solved to optimality. If, however, optimality proving is of less importance, but rather a good feasible IP solution is the major concern, then the information provided by heuristically generated columns can also be used to determine whether or not to stop column generation and start branching.

4.2 Premature termination of column generation

Our column generation scheme exhibits the tailing-off effect, i.e. requiring a large number of iterations to prove LP optimality. Instead of solving the linear program to optimality, i.e. generating columns as long as profitable columns exist, we could end the column generation phase prematurely when the master LP value sufficiently approximates the (theoretical) optimum. Therefore, a lower bound on the master LP is required. Using the information from solving the reduced master and the information provided by solving a pricing problem for each trainee j, it can be shown (see e.g. Hans (2001), Van den Akker et al. (2002); Vanderbeck and Wolsey (1996)) that a lower bound is given by:

$$\delta + \sum_{k=1}^{p} RC_k \theta_k \tag{24}$$

where δ is the objective value of the reduced master, RC_j is the reduced cost of a newly found column for trainee j and θ_j is a binary variable equal to 1 when RC_j is non-negative and set to 0, otherwise. This lower bound is referred to as the Lagrangian lower bound, since it can be shown that it equals the bound obtained by Lagrange relaxation.

4.3 Imbalanced branching

As already outlined in Section 3.2 branching on the x_{ijk} variables is preferred to branching on the z_{jt} variables when optimality proving is a major concern. If, however, fast detection of a good, feasible solution is the main objective, a more imbalanced (and thus more restrictive in one direction) branching scheme like branching on the z_{jt} variables could be more suitable[1]. Indeed, each left branch (z_{jt} set to 1) fixes a full trainee schedule instead of merely a relatively small subset of arcs in the network. Consequently, feasible integer solutions will be detected much sooner. The counterpart is that it will almost be impossible to prove the optimality of a solution (unless the integral solution objective value equals the LP relaxation).

4.4 Combining depth-first and best-first search

Basically, the branch-and-bound tree is traversed in a depth-first way. The advantage of depth-first is that an integer solution is found early and hence upper bound pruning can be applied soon. An important disadvantage, however, is that once an integer solution is found, the algorithm may waste a lot of computation time to improve the solution only slightly. Since we track lower bounds for each node in the search tree, we perfectly know the best possible solution value for each node and all nodes below it. If, upon backtracking, the possible improvement, measured by the difference between the nodes lower bound and the current best found solution, is relatively small, we may opt to backtrack one or more levels further until a node is reached for which the possible gain is worth exploring it. In the extreme case this would be a best-first strategy (i.e. the next node to explore is the one with the lowest lower bound). The disadvantages of a pure best-first search are beside the late detection of an integer solution, the requirement of advanced memory management and sorting capabilities. A mixed approach combining the advantages of both strategies turned out to be a good choice for our application.

[1]To avoid entering the same column twice, each time the column generator discovers a better column, but before updating the incumbent, the new column is checked against the columns in a forbidden list (i.e. set to 0). This can be done quite efficiently since each column can be represented with only four integers using a binary encoding scheme.

4.5 Heuristically fixing x_{ijk} variables

A final heuristic extension involves the fixing of a number of x_{ijk} variables before starting the branch-and-price algorithm. More concretely, a number of 'activity patterns' could already heuristically be scheduled. We refer to an *activity pattern* for activity k from period i_1 until period i_2 as the scheduling of activity k over all time periods between i_1 and i_2 such that exactly one trainee is scheduled at each period. Activity patterns could be identified for all activities for which coverage constraints of type ($=$ or $\geq$) exist. In Beliën and Demeulemeester (2004a) it is shown how a restricted version of the trainee scheduling problem could be completely decomposed on these activity patterns and solved to optimality with column generation. However, as indicated in Beliën and Demeulemeester (2004b), the main disadvantage of this approach is that it could only deal with those trainee-specific constraints which are automatically satisfied when scheduling the activity patterns. Moreover, if (part of) the coverage constraints require two or more trainees to be scheduled, the optimality of a solution could not be proven. The idea is, however, useful to apply in this context. A number of activity patterns could be identified and scheduled heuristically before starting the branch-and-price algorithm. This is done using the greedy heuristic described in Beliën and Demeulemeester (2004a) developed for finding an initial solution before starting their branch-and-price algorithm. Pre-scheduling a number of activity patterns considerably simplifies both the master problem (less coverage constraints) and pricing problem (smaller networks). The more activity patterns are scheduled (i.e. the more x_{ijk} variables are fixed) the easier the problem becomes, but also the less likely we are to find an optimal solution. Upon completion of the branch-and-price algorithm, the best solution is saved as an upper bound and the process restarts with either the scheduling of a different set of activity patterns or a different schedule of the same set of activity patterns.

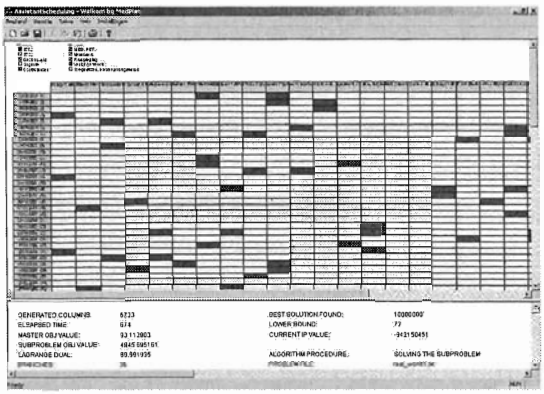
5 Graphical user interface

In this section the graphical user interface (GUI) is presented. The language of the GUI is Dutch. The GUI serves three important objectives.
First of all, it allows for easy data input and constraint specification. Non-available periods for instance are specified by simply double clicking on the corresponding timetable cell and entering the associated penalty cost. The non-available period will be marked in red as indicated in Figure 2(a). Figure 2(c) shows how the properties of a trainee are specified. Each activity having to be performed by the trainee is checked and the target, minimum and maximum number of periods as well as deviation penalty costs and penalty costs for activity split-ups can easily be specified. Figure 2(d) shows an example of a coverage constraint. The corresponding activity, time horizon, type ($\leq, =$ *or* $\geq$), required number of trainees, trainee set (skill category) and penalty costs associated with the coverage constraint have to be specified. The importance of easy, intuitive constraint specification is extremely important for acceptance of the software. If the scheduler would be required to state

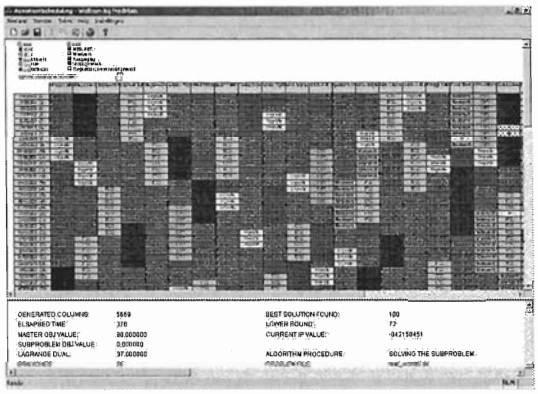
the constraints mathematically (like we did in this paper), it is very likely that (s)he prefers the old manual way of scheduling.

A second objective is the visualization of the search process and of course of the found solution(s). A found solution is represented in Figure 2(b). The visualization of the search process greatly helps to understand how the algorithm works and enables to identify certain problems at an early stage during the search. Figure 3 indicates how the algorithm is visualized during the run. Firstly, each newly found column is drawn. Secondly, each branching restriction is indicated by coloring the corresponding timetable cell with the associated activity color.

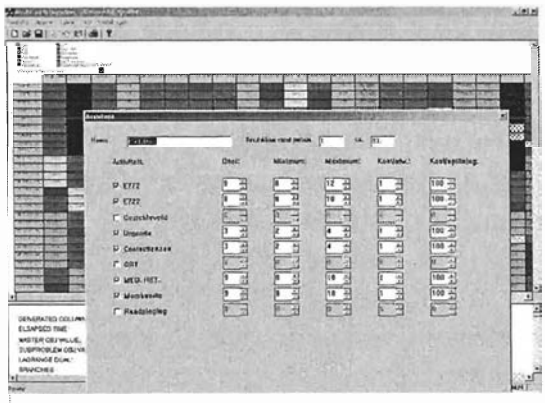
A third objective of the GUI is to enable the user to fix activity assignments before the start of the algorithm and to modify the found schedule afterwards. This can be done very easily by clicking, dragging and dropping. In the extreme case the scheduler can try to build the whole schedule in this (manual) way. If, at a certain moment the scheduler encounters a schedule conflict (e.g. given the partial schedule, in order to satisfy a certain coverage constraint, a trainee would be required to



(a) GUI: non-available periods indicated in red



(b) Visualization of a solution



(c) Specifying the properties of a trainee

(d) Specifying a coverage constraint

Figure 2: The graphical user interface: some screenshots

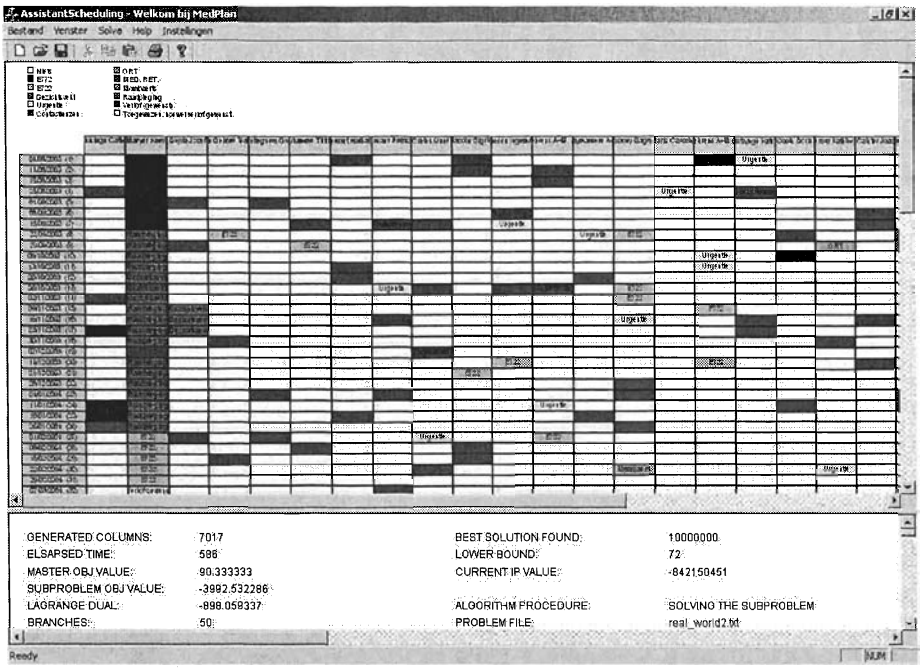


Figure 3: Visualization during algorithm run: the newly found column and the branching decisions indicated in color

restart a certain activity), (s)he can make some assignments undone and run the algorithm to check whether or not a feasible solution is still possible. This feature contributes significantly to the willingness of schedulers to accept the software, since it recognizes the fact that they still have the last word. The software only assists in building the schedules, i.e. it helps in solving difficult 'combinatorial puzzles', but the final decisions are still made by (human) scheduler(s) and not by the PC.

6 Computational results

In this section we present computational results for a real-life trainee scheduling problem encountered at the department *Oogziekenhuis* of the university hospital *Gasthuisberg*, Leuven, Belgium. The number of trainees of this department varies between 20 and 25. These trainees can roughly be divided into four skill categories (depending on academic phase). However, exceptions are possible and occur frequently (e.g. a 3^{rd} year trainee having to perform a 2^{nd} year activity). Schedules are built at the start of each academic year and define the workload for each trainee for all periods in the coming year. Since coverage and non-availability constraints apply on a weekly base and formation requirements are expressed in terms of number of weeks, the basic scheduling unit is a week and thus the number of periods equals 52. In order to simplify the complicated task of the scheduler, current practice includes the aggregation of these 52 weeks in 18 multi-week 'blocks' (16 3-week

blocks and 2 2-week blocks). The disadvantage of this approach is that the scheduler is not able to fully exploit all scheduling possibilities. If, for instance, a trainee is not available during a particular week, then the whole block is made unavailable. Once the schedule is built in terms of these blocks, the remaining week (in case of a 2-week block) or remaining two weeks (in case of a 3-week block) of non-available blocks, are filled up with the scheduling of activities with low set-up costs and for which there is sufficient capacity left. Similarly, formation requirements could not be met to the same level of detail as would be the case if schedules are built on a weekly base. Consequently, the resulting schedules were frequently observed as being unfair and had to go through an extended bargaining process. The total time needed to build the schedule, bargain, rebuild etc... could easily take about 10 days for an experienced scheduler.
Merely for illustration purposes, we provide computational results for the 2003-2004 academic year trainee scheduling problem. Firstly, we consider the 18-blocks problem, which could be solved rather easily. Next, we try to solve the 52-weeks problem, which is much more complicated, but allows for the construction of more detailed and qualitatively better schedules for the same problem. For this last problem, we show how the heuristic extensions can help finding a good (better) solution in less time. Table 2 summarizes the most important properties of both problems.

Table 2: Real-life problem

Problem	Nr. of periods	Nr. of trainees	Avg. nr. of activities for each trainee	Nr. of coverage constraints in the master[a]
1	18	21	6	260
2	52	21	6	720

[a] Exclusive the constraints removed by constraint preprocessing (=+/- 10% of total number of constraints).

All our experiments were performed on a 2.4 GHz Pentium 4 PC with the Windows XP operating system. The algorithm was written in MS Visual C++.NET and linked with the CPLEX 8.1 optimization library. Computational results are given in Table 3. The first line of this table indicates that the 18-period problem could be solved to optimality within 2205 seconds. The gap with the optimal LP relaxation is 2%. The gap is defined as $100 * (solution - LP\_relaxation)/(LP\_relaxation)$. The next lines illustrate the impact of the different heuristic extensions on the solution quality. The second column indicates the section numbers of the implemented heuristic extensions. As can be observed, these extensions are implemented in a cumulative way. The computation times were limited to 300 seconds. If we apply the heuristic instead of the exact column generator, the same (optimal) solution was detected. We note that optimality was not proven since (a) the LP relaxations are not proven to be optimal and (b) the branch-and-bound tree still contained unexplored nodes. If we allow for an LP optimality gap of 2% in each node of the branch-and-bound search tree the best solution found had a gap of 3.5%. Next, if

the balanced branching scheme is replaced with an imbalanced one (branching on the column variables) the gap increases to 5%. However the time needed to find the first integer solution was decreased significantly with almost 50% (from 40 to 23 seconds). The problem here is that the algorithm wasted a lot of time exploring nodes below a right branch (a particular column variable fixed to 0) only to improve the solution slightly. Strong branching decisions (column variables fixed to 1) made near the root of the search tree could not be made undone within the restricted time limit. When the depth-first way of search was combined with a best-first search by requiring a minimal possible improvement for exploring a node (as outlined in Section 4.4), a better solution was detected. Finally, if a number of activity patterns are scheduled heuristically (as outlined in Section 4.5) an integral solution was already found in less than 10 seconds, however the algorithm lacked the flexibility to find close to optimal solutions. In the first setting (*) two activity patterns were identified and scheduled heuristically, whereas in the second setting (**) four activity patterns were pre-scheduled (freezing approximately 10% and 20% of the schedule). Afterwards the branch-and-price algorithm was run to solve the remaining problem to optimality. Upon completion of branch-and-price the process restarts with a different activity pattern set and/or a different scheduling of the same set. In many cases the branch-and-price algorithm could be terminated as soon as the LP lower bound exceeded the current best found solution. For this relatively small problem, the results indicate that the gaps tend to increase with added heuristic extensions, however the time needed to find a first integral solution decreases.

Let us now turn to the 52-period problem. As indicated in the second line of Table 3, this problem could not be solved to optimality by the exact branch-and-price algorithm (i.e. without heuristic extensions) within 10 hours of computation time. There was a gap of 18.66% between the best solution found and the optimal solution of the LP relaxation. Since only a relatively small number of nodes in the branch-and-bound tree was explored (254) and we know from experience that the LP relaxation gap is usually much smaller, there is strong indication that better solutions should be possible. Again, we implemented several heuristic implementations and report on the gaps found. The computation times were limited to 900 seconds. When we replaced the exact column generator with a heuristic algorithm (Section 4.1), the gap was reduced with 5.3% (from 18.7% to 13.4%). Taken into account the restricted computation time many more nodes could be evaluated in the search tree (111 nodes in 900 seconds compared to 254 nodes in 36000 seconds). The main reason for this improvement is the fact that the algorithm suffered less from the so-called 'tailing-off effect' observed in many column generation applications. Tailing-off means that the algorithm keeps finding columns with negative reduced cost, but these columns fail to improve the LP objective. In other words, upon LP convergence, many columns are added merely to *prove* LP optimality, but do not result in a decrease of the LP objective. Recall that also in the exact algorithm the main part of the columns were generated with the heuristic column generator. Only those needed to prove LP optimality had to be generated using the exact column generator. The same reasoning applies if we allow for an LP optimality gap of 2%

in each node of the branch-and-bound search tree and the solution could be further improved until 7.3% of the LP relaxation.

Table 3: Computational results

Problem	Heuristic extensions[a]	Time LP relaxation (s)	Time first integer solution (s)	Total computation time (s)	LP relaxation gap	Nr. nodes	Nr. columns
1	-	21	56	2205	2.0%	534	22184
1	[4.1]	17	45	300	2.0%	122	5541
1	[4.1],[4.2]	16	40	300	3.5%	143	5126
1	[4.1],[4.2],[4.3]	16	23	300	5%	312	6265
1	[4.1],[4.2],[4.3],[4.4]	16	23	300	3.1%	278	5963
1	[4.1],[4.2],[4.3],[4.4],[4.5]*	3	10	300	7.6%	34	2497
1	[4.1],[4.2],[4.3],[4.4],[4.5]**	3	8	300	12.3%	25	1842
2	-	184	922	36000	18.7%	254	28515
2	[4.1]	111	646	900	13.4%	111	9456
2	[4.1],[4.2]	105	574	900	7.3%	154	11414
2	[4.1],[4.2],[4.3]	105	324	900	9.6%	377	12438
2	[4.1],[4.2],[4.3],[4.4]	105	324	900	5.6%	316	12723
2	[4.1],[4.2],[4.3],[4.4],[4.5]*	26	46	900	11.9%	123	8456
2	[4.1],[4.2],[4.3],[4.4],[4.5]**	4	18	900	15.3%	88	7458

[a] The section numbers of the implemented heuristic extensions are indicated. In premature termination (4.2) column generation is ended if the LP optimality gap is smaller than 2%. In imbalanced branching (4.3) branching occurs on the column variables instead of on the timetable cells. In partial best-first search (4.4) a minimal possible improvement of 2% is required for exploring a node in the search tree. In heuristic fixing [4.5]*, two activity patterns are pre-scheduled, freezing approximately 10% of the schedule, whereas in heuristic fixing [4.5]**, four activities were pre-scheduled, freezing approximately 20% of the schedule.

Next, if the balanced branching scheme is replaced with an imbalanced one (branching on the column variables) the algorithm was not able to find a better solution. However the time needed to find the first integer solution was again decreased with almost 50% (from 574 to 324 seconds). When the depth-first way of search was combined with a best-first search, a better solution could be detected. Finally, if a number of activity patterns are scheduled heuristically an integral solution was already found in respectively 46 and 18 seconds, but the gaps increase again.

7 Conclusion

The problem of building long term trainee schedules has been studied. A decomposition approach has been applied to solve the problem. Therefore, the problem has been reformulated in terms of decision variables which explicitly satisfy all trainee-specific constraints. Column generation could be applied to find the LP relaxation of this new formulation and a branching scheme has been proposed to drive the solution into integrality. Next, it has been shown how this approach can easily be turned into an effective heuristic algorithm. Therefore, five heuristic extensions have been proposed. A graphical user interface (GUI) has been developed. The GUI allows for easy data input, constraint specification and modification of the algorithmic

settings. Moreover, certain parts of the schedule could be frozen before the algorithm is run and proposed solutions could be easily modified. A real-life problem has been solved with the developed application and computational results are given. These results illustrate how the different heuristic extensions could improve solution quality if the problem is too complex to find a (proven) optimal solution.

Acknowledgements

We acknowledge the support given to this project by the Fonds voor Wetenschappelijk Onderzoek (FWO) - Vlaanderen, Belgium under contract number G.0463.04. We are grateful to Prof. Dr. W. Sermeus and Kris Vanhaecht of the Centrum voor Ziekenhuis- en Verplegingswetenschap (CZV) for drawing our attention to challenging scheduling problems in health care management and to Jenny Cristael (Oogziekenhuis UZ Leuven, Belgium) for providing case study data.

References

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P. and Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs, *Operations Research* **46**: 316–329.

Beaumont, N. (1997). Scheduling staff using mixed integer programming, *European Journal of Operational Research* **98**: 473–484.

Beliën, J. and Demeulemeester, E. (2004a). Scheduling trainees at a hospital department using a branch-and-price approach, *Research Report OR 0403*, Katholieke Universiteit Leuven, Department of Applied Economics.

Beliën, J. and Demeulemeester, E. (2004b). On the trade-off between staff-decomposed and activity-decomposed column generation for a staff scheduling problem, *Research Report OR 0412*, Katholieke Universiteit Leuven, Department of Applied Economics.

Cappanera, P. and Gallo, G. (2001). A multi-commodity flow approach to the crew rostering problem, *Technical Report TR-01-08*, Dip. di Informatica, Univ. di Pisa.

Caprara, A., Monaci, M. and Toth, P. (2003). Models and algorithms for a staff scheduling problem, *Mathematical Programming* **98**: 445–476.

Hans, E. W. (2001). *Resource loading by branch-and-price techniques*, Ph.D. dissertation, Twente University Press, Enschede, The Netherlands.

Mason, A. J. and Smith, M. C. (1998). A nested column generator for solving rostering problems with integer programming, *International Conference on Optimisation: Techniques and Applications*, pp. 827–834.

Mehrotra, A., Murphy, K. E. and Trick, M. A. (2000). Optimal shift scheduling: A branch-and-price approach, *Naval Research Logistics* **47**: 185–200.

Van den Akker, M., Hoogeveen, H. and Van de velde, S. L. (2002). Combining column generation and lagrangian relaxation to solve a single-machine common due date problem, *INFORMS Journal on Computing* **14**: 37–51.

Vanderbeck, F. (2000). On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm, *Operations Research* **48**: 111–128.

Vanderbeck, F. and Wolsey, L. A. (1996). An exact algorithm for IP column generation, *Operations Research Letters* **19**: 151–159.

Warner, D. M. (1976). Scheduling nursing personnel according to nursing preferences: A mathematical programming approach, *Operations research* **24**: 842–856.